Multimed Tools Appl manuscript No.
(will be inserted by the editor)

Automatic Terrain Generation with a Sketching Tool

Anna Puig-Centelles · Peter A. C. Varley · Oscar Ripolles · Miguel Chover

Received: date / Accepted: date

Abstract Nowadays, applications such as scientific simulations, virtual reality or computer games are increasing the detail of their environments with the aim of offering more realism. Terrain is a very important element in these outdoor scenarios. Artists are currently requiring a higher level of customization. In this sense, the objective of the work presented in this paper is to provide the final user with an easy-to-use terrain generation application. More precisely, our aim is to ease the creation of islands. We propose a sketching solution which, combined with a simple terrain algorithm, is capable of suiting the user needs. The application is composed of two windows, which offer 2D and 3D representations of the terrain respectively. These windows are sufficient for providing the user with an interactive feedback about the island that is being designed. We try to show that relatively simple algorithms can be combined to provide successful results.

Keywords Terrain Generation · Sketching Inteface · Virtual Environment · Simulation

1 Introduction

Terrain generation is a research area which has been active for many decades and growing power of modern computers has made them capable of producing increasingly realistic scenarios. Synthetic terrain generation is a process which creates elevation values throughout a two dimensional grid. The need for highly realistic scenarios often involves developing algorithms that can generate more realistic terrains with more user control over the final terrain that is created.

A. Puig-Centelles · P. A. C. Varley · M. Chover
Institute of New Imaging Technologies, Universitat Jaume I
Av. Vicent Sos Baynat s/n, Castellon, Spain
E-mail: apuig@uji.es, varley@uji.es, chover@uji.es

O. Ripolles
Inst. Universitario de Automatica y Informatica Industrial, Universidad Politecnica de Valencia
Camino de Vera s/n, Valencia, Spain
E-mail: oripolles@ai2.upv.es

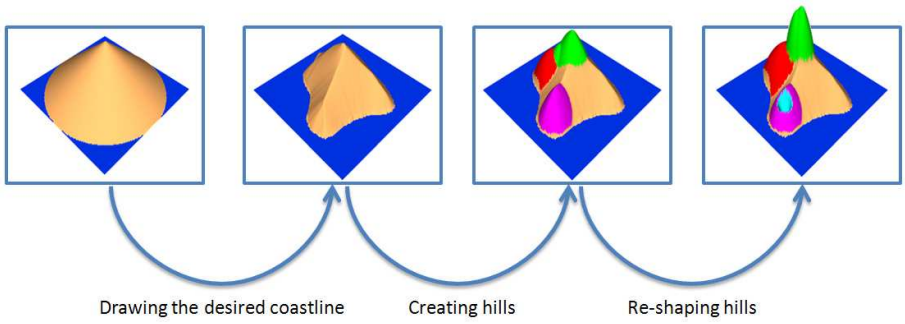


Fig. 1 Designing a sample island terrain in three steps.

The software community has commercialised many software tools that can offer incredibly realistic environments. However, these software applications are usually very difficult to use and the artist is presented with a large amount of tools to model the terrain. Therefore, usability is a key element and sketching is a very promising solution that can simplify the interface while producing satisfactory terrain.

Sketching is a very promising tool for terrain generation. The sketching applications can offer a great amount of user control for the terrain synthesis process. Sketching is commonly understood as the process of rapidly executing freehand drawing where the obtained sketches are not considered to be finished work. In the specific case of computer-aided modeling, sketching on a piece of paper is often used in the prototyping stage, before an experienced 3D modeler converts these ideas into a 3D model by means of specific software like 3D Studio Max [14].

Sketching is a tool that is well suited to the design of architectural elements and it provides the user with a considerable amount of control over the created elements. Research has produced prototype tools for interpreting sketches of abstract polyhedra [3,15], and even for extracting models of buildings from photographs [12]. It is possible to find recent surveys on sketching [23], where the reader can account for the great amount of work that has been developed in this area. However, less work has been focused on sketching the underlying terrain or extracting it from a photograph. Thus, buildings are often considered as the *foreground* and are taken into account properly, whereas terrain is seen as *background* which is often ignored.

Our aim consists in developing convenient and simple ways to create computer models of terrain. Our work is oriented toward novice users by offering rapid prototyping and rapid content creation, in contrast with those solutions which offer slow creation of professional-quality graphics. Obviously, the more features a program has, the more complicated the user interface is. Most CAD applications have extremely complex user interfaces (typically with over 50 buttons and over 50 menu options) and they are very hard to use, particularly for beginners. In this sense, including many features is a positive value, but complex user interfaces are negative. On the contrary, some solutions can be considered as in the other extreme [4,35], with the minimum possible user interface. These applications are

Fig. 2 Terrain obtained with our framework and imported into the Torque Game Engine.

also difficult to use, because the user does not have enough control over what is happening.

Consequently, the optimum is somewhere in between these two extremes (high user control and low complexity of use), and our application has been developed as an attempt to fulfil both requisites. Our goal is similar to the idea given in [19], where the authors show that relatively simple algorithms can provide non-professional users with fast, successful results. It is worth mentioning that, more precisely, in this paper we address the problem of creating computer models of islands for use in computer games. Terrain sketching applications commonly present the surfaces floating in mid-air as a patch, without continuing to the horizon in the manner of a landscape. In our tool we focus on the generation of islands, so that the terrain is surrounded by water. In this sense, Figure 1 shows how we can create a terrain with three simple steps using the method we are presenting. Thus, we can sketch the island, create some hills and also re-locate and re-shape them as desired. Figure 2 presents a terrain generated with our solution and imported into a game engine for its use in a 3D scenario. In the following sections we will describe how this terrain is generated automatically from the user's input.

In this work we describe a terrain creation algorithm for island terrains based on *heightmaps*, which are regularly-spaced two-dimensional grids of height coordinates. These grids can be later processed by a modeling software or game engine to obtain the 3D surface of the desired terrain. The elevation of the terrain is automatically calculated from the coastline sketched by the user, who can also create hills and apply filters in order to achieve more irregular terrain. The perturbations created by the hills and the filters are both created by means of a simple yet efficient terrain generation algorithm. Once again, it is important to mention that our aim is to provide the user with more control over the terrain appearance and over the placement and shape of terrain features.

Furthermore, we also consider the integration of our algorithm into a sketching application. This application combines a 2D representation window and a 3D displaying window in order to simplify the drawing process. The use of these two windows to model the terrain offers many advantages and simplifies the modeling

experience. First, the user creates a silhouette of the island in the 2D window and, then, the user will be able to modify the terrain appearance from the 2D and the 3D windows. The terrain thus obtained will be outputted as a heightmap that may then be imported into a game engine.

This work is organised as follows. Section 2 contains the state of the art in terrain generation techniques as well as sketching freeform surfaces. Section 3 describes our terrain generation algorithm. After that, Section 4 analyzes our sketching application and presents the user interface and the possible operations that can be used to create the terrain. In Section 5 we describe a detailed implementation of the data structures and processes of our framework. Later, Section 6 depicts our results and it also discusses a usability test performed among different potential users. Finally, Section 7 presents our conclusions and gives some ideas for future work.

2 Related Work

In this section we analyze and characterize the different approaches that currently exist for terrain generation. After that we will consider the different software tools which are available for creating artificial terrain. Finally, we will give some basic ideas on sketching and its application to our purposes.

2.1 Terrain Generation

It is possible to find in the literature a wealth of research on synthetic terrain generation. When generating artificial terrain, the techniques can be grouped into three different categories:

2.1.1 Procedural Approaches

This category gathers those methods that generate the terrain automatically. These methods can be further separated into:

- **Fractal Landscape Terrain Generation**. These are the most popular procedural approaches [30], as they are efficient but difficult for users to control. It is possible to find a review of recent fractal approaches in [6].
- **Physical Erosion Simulation**. These techniques simulate the effects of physical processes such as erosion by streams [16], water [20] or wind [37]. A recent technique that combines a non-expensive fluid simulation with an erosion algorithm is presented in [21]. It also supports effects like dissolving, transportation and sedimentation of material in the process of erosion.
- **Genetic Algorithms**. These methods use some initial input to produce a variety of terrains by means of genetic algorithms. The solution proposed in [24] employs a two-pass approach which is efficient but very difficult for a user to control. In order to improve on genetic solutions, Frade et al. [9] developed a Genetic Terrain Programming approach which allowed users to *evolve* their terrains under some aesthetic concept or desired feature.

The main problem of these different techniques is the fact that modifying their parameters to obtain a desired terrain may be a painstaking task.

2.1.2 Real Terrain Information

This approach groups the techniques from the Geographic Information Systems (GIS), where elevation data come from real-world measurements [39]. GIS data can be acquired from a number of sources and in different formats, such as the U.S. Geological Survey's Digital Elevation Model (DEM) format [34]. Some authors use contour lines of terrain to reconstruct the surfaces by interpolating the values between the different lines [13]. Similarly, another possible source of information could be the study of the extraction of terrain from photographs [8].

All these approaches have the advantage of offering highly realistic terrains in very little time, but with little user control.

2.1.3 User Defined Approaches

This is the most flexible type of techniques, in which a human artist creates the terrain manually by using different kinds of software, like 3D modeling software, sketching techniques, specialized terrain editor programs or the editors that are included in many game engines. The complexity of these techniques depends on the complexity of the selected software itself.

2.2 Terrain Software

Among the user-defined approaches, specific terrain generation software has received a great attention from the modeling community. In this section we introduce some terrain tools for simulating artificial environments. In these applications the user sets parameters and the program creates a pseudo-random landscape which meets those parameters. In all of these programs, terrain is modeled and imported/exported as a heightmap.

Among the existing software for terrain synthesis we highlight:

- Terragen [26], which has evolved from a terrain generator to an application with complex atmospheric effects, HDR lightning and includes the possibility of importing your own objects. These features enable the user to obtain complex environments which can also be animated in time.
- Grome [27] which, like Terragen, uses a procedural creation of geometry. This software also includes natural erosion, procedural textures and the possibility of working with layers to simplify the user experience.
- L3DT [33] is another software application that generates artificial heightfields and exports their data to multiple formats including the terrain format used by TGEA [11,18].
- Terraineer [31] offers the possibility of experimenting with different height generation algorithms.
- Worldbuilder [7], which includes the possibility of adding vegetation, different water effects and complex materials, offering very realistic final renders. Moreover, the authors provide the user with an easy-to-use elevation editor.
- World Machine [29] is based on fractals and a complex graph system to organize the different elements that give form to the terrain. In this sense, this software includes the modeling of physical weathering processes like wind, water and other natural processes.

All of these systems have evolved in the latest years and are capable of offering very realistic terrains. Nonetheless, this increase in visual quality also involves an increase in the complexity of the application.

2.3 Sketching terrain

Sketching has been commonly applied to model general 3D objects, although there have been proposals to model natural elements like plants [1], clouds [38] or terrain [36, 10].

The Harold system [4] was an initial approach for sketching-based modeling. The basic idea was to create a 3D world with the sketched information but populating it with the sketches themselves. In this sense, the process is similar to an image-based solution and the final image is equal to the sketched one, although it allows the user to change the viewpoint and interact with the scenario. With this objective the main primitive of the Harold system was the billboard in which the strokes were stored. Regarding terrain, the sketched lines of the user define bumps and hills that are considered to lift the affected objects. This multi-modal system was very promising, although the scenes observed from a different viewpoint usually included strange artifacts. Nevertheless, their objective was not realistic rendering as they aimed at capturing a child's drawing into a 3D world.

Later, Watanabe et al. [36] proposed a Java-based application in which the user could use simple line strokes to create the mountains. Moreover, the user could add noise, textures or rivers to offer a more realistic appearance. This initial work offered some promising results although the terrain totally lacked realism and the operations allowed by the software were very limited.

The work developed by Zhou et al. proposed a solution from a totally different perspective [41]. In their approach patches from sample terrain (obtained from a DEM) were used to generate new terrain and the synthesis was guided by a user-sketched feature map that specified where terrain features occurred in the resulting synthetic terrain. Although the results were very realistic, the user implication in the finally obtained terrain was limited and complicated. The beautiful images offered by the authors were obtained with Terragen [26], although later the proposal was implemented as a plug-in to the World Machine commercial application [29].

Following a similar technique, Belhadj [2] presented in the same year a method for reconstructing terrain from Digital Elevation Models. His approach uses fractal-based algorithms for performing the reconstruction and enables the users to sketch details on the terrain or create a new model from scratch using an image editor.

Varley et al. [35] introduced a simple interface for sketching heightmaps of islands. This application was very simple but, although it offered a good amount of user control, it was difficult to use and the obtained terrain was not completely customizable.

More recently, Gain et al. proposed a new sketching application for terrain modeling [10]. In this solution the authors offer the users the possibility to sketch the silhouette of the heights of the mountains and also the area of influence of this silhouette, widening or stretching the mountains. They also developed a fast multiresolution surface deformation so that the mesh representing the terrain can adapt to those areas where the surface is more detailed. Nevertheless, the proposed solution still requires a complex interaction from the user where multiple views of

Authors	User control	Complexity of use	Features controlled	Quality of terrain
[4]	High	High	Mountains, trees, multiple objects	Low
[36]	Medium	Low	Mountains, rivers, colors, noise	Medium-low
[2]	Medium-low	High	Hills, rivers	Medium-high
[41]	Medium-low	High	Mountains, rivers, valleys	High via Terragen
[35]	Medium	Medium	Silhouette, height of mountains	Medium-low
[28]	Low	High	Mountains	High via Terragen
[10]	Medium-high	Medium	Mountains, noise	Medium-high
[32]	Low	Medium-low	Ridges	Medium
[40]	Medium-Low	Medium-high	Heightmap	Medium
Our proposal	Medium-high	Medium-low	Mountains, craters, silhouette, noise	Medium-high

Table 1 Characterization of applications for sketching terrain.

the terrain are needed in order to obtain the desired terrain. Moreover, intersecting mountains can become difficult to work with as the user must decide if a new mountain must be in front or behind an existing one while drawing the silhouette.

Based on the graph theory, the method proposed by Rusnell et al. [28] uses user-drawn strokes to define the main features of the terrain and applies *path planning* to generate the terrain. The control over the obtained terrain is slightly limited and the lack of feedback prevents the user from modifying the sketch easily. Nevertheless, they present nice visual images by using the Terragen software [26].

From a different perspective, the work presented in [32] proposes a simple application where sketching some initial ridge lines is sufficient to guide the terrain generation. Thus, using this information, the application generates a rivers system where meanders and tributaries are considered. Once the river network is constructed, mountains are calculated accordingly. Although the system is very easy to use, the control over the generated terrain is very limited.

With the aim of offering a complete solution, Wünsche et al. [40] introduces a framework where the user can sketch terrain, vegetation and *blobby* objects. In a later stage, the developed scenarios are enriched with multimedia information. The terrain generation is based on sketching contour lines and arrows indicating gradient, being the system in charge of triangulating the surface and calculating heights according to the gradients indicated. An important limitation of this proposal is that it is not possible to interactively alter the sketches while seeing the resulting geometry. The authors acknowledge themselves that developing a unified solution to sketch and visualize environments remains an open line of work.

Table 1 offers a qualitative characterization of the latest techniques for sketching terrain. Our proposal has also been considered in this characterization. The description of each solution presented includes the different aspects that we detail next:

– User control: this value offers an estimation of the amount of user control offered by the application.

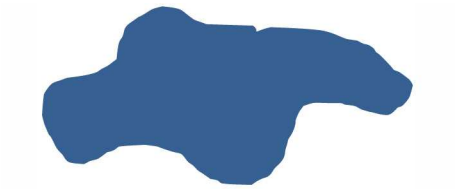


Fig. 3 Sample silhouette of an island.

- Complexity of use: this indicates how complex the sketching operations are.
- Features controlled: it indicates the different terrain features that can be sketched.
- Quality of terrain: this offers an estimation of the overall quality that can be obtained, considering the features controlled and the example terrains offered in the research papers. Image quality is not considered as many proposals resort to professional software for the final renders.

Most of the proposals that have been reviewed do not provide enough user control. Many of them use some previously-made sketch of the terrain, while only a small subset (papers [4,36,35,10]) allows the user to start from scratch and to interactively sketch the terrain while visualizing the result. Nevertheless, these latter applications are still difficult to control using the different parameters and operations available. In this sense, we believe that there is an increasing demand for more intuitive interfaces, as many designers and more general users are often disappointed by the complexity, difficulty and unintuitive nature of current modeling interfaces. It is our aim to develop a terrain generation technique which is simple yet efficient.

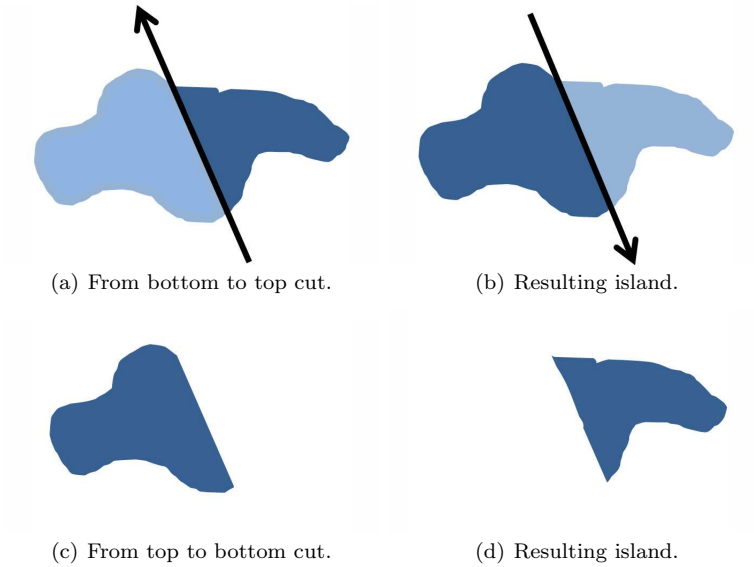
3 Our Terrain Generation Algorithm

The method that we present for generating islands is based on the use of heightmaps and it allows users to define and modify the coastline of the island, using an image as guide if desired. Furthermore, they can also create and reshape any number of hills, which will interact with each other and with the existing terrain. Finally, we offer the possibility of applying filters to give the final terrain a more realistic appearance.

3.1 Reshaping the Coastline

The user can draw the silhouette of the island freely, but it will be necessary to delineate a continuous curve by sketching a closed shape, as shown in Figure 3. The shape of the coastline can be changed by redrawing, starting and ending at points near the existing coastline and drawing a continuous curve in any direction between those two points. Thus, it is possible to apply different operations in order to modify the existing coastline.

The user may decide to cut a piece of the island off. As a consequence, the terrain will be split into two areas. Depending on the direction of the cut, the algorithm will decide which one of these areas is to be rejected. The direction of



(a) From bottom to top cut. (b) Resulting island.

(c) From top to bottom cut. (d) Resulting island.

Fig. 4 Cutting and reshaping the island.

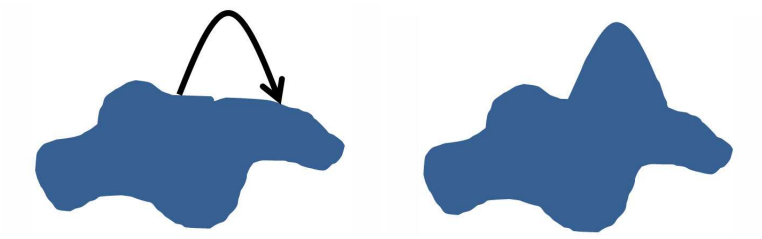
the cut is being understood as the direction running from the start to the finish point. The rejected area will be the one on the left-hand side of the direction of the cut. Figure 4 presents an initial coastline and the silhouettes obtained after performing cuts with the same start and finish points but with different directions.

Furthermore, the user can also add new pieces to the existing area. Again, the algorithm will behave differently depending on the direction of the sketched draw. If the line has been sketched clockwise, the *new* area will be added to the existing one. In contrast, if the line has been performed in an counterclockwise direction, then this *new* area will be maintained as the *new* area and the *old* one will be rejected. In Figure 5 we can see an example of these possible ways of modifying the area by adding or subtracting a piece of terrain.

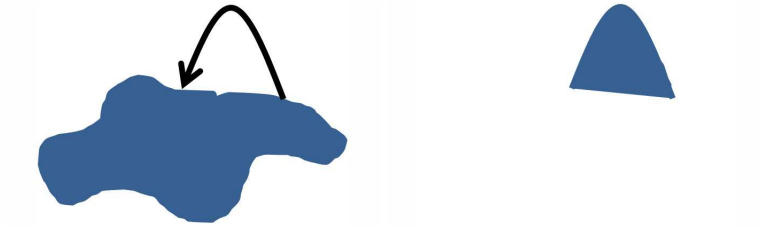
We must note that the algorithm will differentiate between cut and supplement operations by testing whether the line goes through the terrain area or not. In those cases in which a line is used to perform more than one operation, each point where the line intersects the silhouette will be interpreted as the finish and start points of the consecutive operations. In Figure 6 we depict an area that is being modified by two consecutive operations: the first one consists in an external clockwise supplement and the second one is a curved internal cut that is rejecting the piece of its left-hand side.

3.2 Updating the Terrain Height

Every time the coastline silhouette is modified, the terrain algorithm has to respond accordingly to those changes and recalculate the height of the terrain in order to offer a continuous surface.

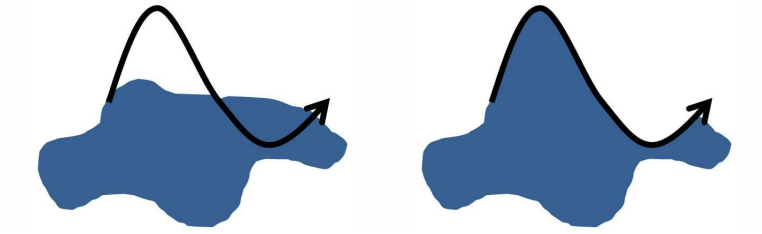


(a) Clockwise supplement. (b) Resulting island.



(c) Anticlockwise supplement. (d) Resulting island.

Fig. 5 Supplementing and reshaping the island.

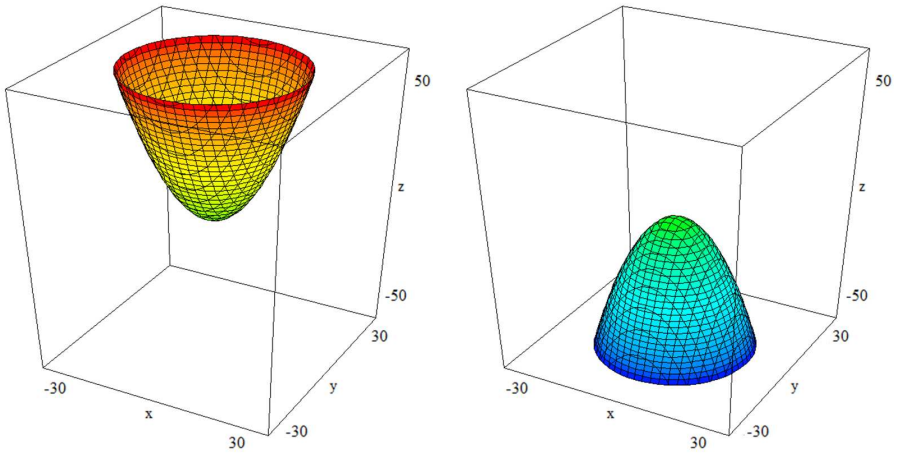


(a) Combined operations. (b) Resulting island.

Fig. 6 Two consecutive operations.

Since we are simulating the terrain of an island, we must take into account the level of the sea. As a consequence, when the coastline changes, it may be necessary to modify the elevations of some onshore points. Ideally, points close to *old* parts of the coastline which remain unchanged should also remain unchanged, but points close to *new* parts of the coastline should be elevated above sea level regardless of their previous height. Therefore, if we reshape the coastline then we have to check whether all the points contained inside the island have the appropriate height.

In order to obtain the new height values, we take into account the distance from each point to the nearest piece of *new* coastline D_n and to the nearest piece of *old* coastline D_o, both scaled to the range 0 to 1. The height of each onshore point H_i is calculated as a weighted value between the *old* height H_o and the *new* one H_n, the latter being proportional to D_n. We implement the calculation of the heights with Equation 1,where the weight W is calculated by Equation 2.



(a) Elliptic paraboloid upwards. (b) Elliptic paraboloid downwards.

Fig. 7 Example view of elliptic paraboloids.

$$H_i = H_o(1.0 - W) + H_n W \tag{1}$$
$$W = 0.5 + 0.5tanh((D_o)^2 - D_n) \tag{2}$$

The hyperbolic tangent ($tanh$) function is chosen because it has the appropriate shape, which is close to -1 for points near the *old* coastline and close to 1 for points near the *new* coastline. Moreover, it never goes outside this range. D_0 is squared so that points close to neither coastline are treated as being closer to *old* rather than to *new* coastline.

Whether a pixel is onshore or not is assessed by referring to a silhouette of the island which is recalculated after each change to the coastline. The process entails drawing the coastline on a blank array of pixels and using a flood-fill routine (starting from a point clearly outside the coastline) to distinguish sea from land.

3.3 Generating Hills

In our work we want to ease the modification of the orography, which refers to the relief of mountains, hills and any other elevated region of a terrain. The idea is to allow the user to create multiple hills having the desired radii, height and location over the terrain.

In our algorithm we define hills as *elliptic paraboloids*. An elliptic paraboloid is shaped like an oval cup. In a suitable coordinate system, it can be represented by the equation:

$$\frac{z}{c} = \frac{x^2}{a^2} + \frac{y^2}{b^2} \tag{3}$$

considering that the *elliptic paraboloid* is centred on $0, 0, 0$ with radius a, b, c (along the x, y and z axes), being $a, b, c \in \Re$ and $a > b$. The variable raised to the first power indicates the axis of the paraboloid, in our case z.

This function represents an elliptical paraboloid which opens upwards and can be seen in Figure 7b. A negative value of c defines an elliptical paraboloid which opens downwards. This latter quadratic surface will be used in our algorithm to define the hills. It is important to note that we allow the user to define valleys, lakes or craters by means of elliptical paraboloids which open upwards as can be seen in Figure 7a.

With this equation the user can introduce the central point, the radius and the height of each hill. Once we have this information, our algorithm will be able to calculate the height of each point affected by the hill. All those points are obtained with the central point and the radius that have been defined. The height of each single point will be modified by following Equation 3. More precisely, we will add the new height to the previous one. In this sense, if we combine several hills the heights will be added. By so doing, we allow for the creation of valleys and volcanic mountains. As an example, in the results Section Figure 14(a) presents a volcano, which is created by locating two hills at the same place and modifiying one of them to have a negative height. Thus, by adding the height values of both hills we obtain the expected surface.

3.4 Filtering the Terrain

In order to obtain a better appearance for the terrain being designed, we can introduce some fuzzy bumps to deform the regular surface. We have implemented a filter to introduce *noise* into the previously defined rounded terrain. This filter can be applied as many times as the user desires and it will give us a number of perturbations proportional to the surface area of the island. The perturbations will be randomly distributed throughout the island surface. These perturbations will also have an elliptical paraboloid shape, but they will be wider than taller and they will be produced upwards or downwards in a stochastic manner. The height (positive or negative) of these perturbations is stored and added to the heights of the terrain surface, so that if we modify the terrain the perturbations are applied to the newly created terrain.

Figure 8 shows how the perturbations are located all around the island. The green bumps are the small hills and the red ones are the small valleys that will deform the regular surface of the terrain. It is worth mentioning that more complex procedural noise may have been applied, although the proposed approach is sufficient for our objectives. Due to the hand-made production of the hills and the random filters that are applied, it is very unlikely to obtain two identical terrain surfaces.

4 Our User Interface Based on Two Windows

This section describes our sketching application for terrain generation by using the ideas presented earlier. The proposed application tries to maintain the three basic elements that form the traditional sketching on a piece of paper:

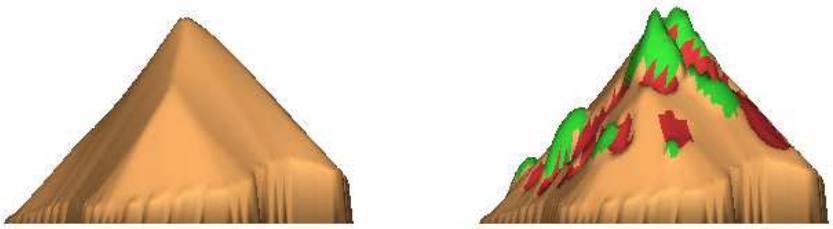
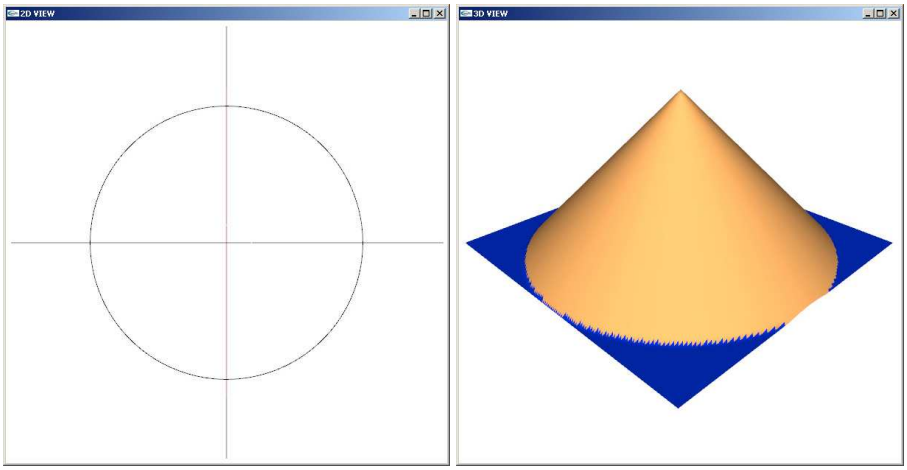


Fig. 8 Filter applied to the terrain. The green bumps represent the small hills and the red ones the small valleys.



(a) 2D window. (b) 3D window.

Fig. 9 2D and 3D Window on Startup.

- feedback, enabling the artist to visually compare the improvements on the sketch.
- re-sketching, allowing the user to modify the previous appearance of the object.
- incremental refinement, adding detail to the object until the artist is satisfied.

These three elements are fundamental for the proper use of a sketching tool [5] and have been addressed in our proposal in order to assure that the final user have a correct sketching experience.

Our proposed framework offers the user an interactive sketching application. This solution consists of two windows, which is key to simplify and improve the sketching process. The 2D window depicts the silhouette of the coastline of the island, as seen in Figure 9a. The 3D window represents the volumetric view of the whole island, as seen in Figure 9b. This 3D view presents a surface which is automatically constructed with the information stored in the heightmap. When the implementation starts, the 2D window contains a circular coastline, as shown in Figure 9a. The 3D window, shown in Figure 9b, depicts a conical island, which is the initial terrain that the user will be able to modify.

The presented interface is close to the ideal of a modeless single-tool interface, with all of its major operations being controlled by a single device (pen or single-button mouse). A problem that appears in some of the most advanced sketching applications, like [25], is that they require a multi-modal push-button interface. Our intention is to maintain the original sketching objectives in order to keep our application as simple and natural to use as possible. Nevertheless, it has been necessary to develop a two-button mouse software to integrate all the functionalities presented in the previous section. In the following subsections we will provide a detailed description of the interaction with the aforementioned windows in our application.

4.1 2D Modeling Operations

The 2D window allows the user to perform two basic sketching operations: defining the silhouette of the island and adding and modifying hills.

When interacting with the left mouse button in this window, the user is allowed to design the coastline of the island. It is possible to draw a free-form silhouette interactively and the system will simultaneously update the terrain. As we have mentioned in previous sections, it is possible to cut and extend the existing terrain by defining lines that start and end on the coastline. In this way, we could draw any irregular shape to delimit the terrain of our island. Our system includes an additional feature, which has proved popular with users: when a change is made to the coastline, the *old* coastline gradually fades away, taking about two seconds to do so. Pressing the right mouse button during this period removes the amended coastline and reverts back to the *old* coastline.

In our application, when the user clicks with the right button either inside or outside the coastline, the application understands that the user is defining the central point of a hill. Then a colored circular line will appear on the terrain surface surrounding the central point that has just been created. This line represents the area influenced by that particular hill. The user may add as many hills as desired and each one will be depicted in a different color. It is important to comment that when defining hills either inside or outside the coastline, both will affect the terrain that has been generated since both radii affect the coastline. After defining the hills, the user will be able to modify the radii and the location of the hills inside the island:

- Right-clicking on the centre point of the hill and dragging, allows the user to change the position of the hill. The user can eliminate a hill by dragging it outside the island until its radius is completely outside the coastline.
- Right-clicking and dragging on the circular line allows the user to modify the hill radius.

The application includes the possibility of *zooming* in on the sketched island by clicking the right button of the mouse. Using the zoom can help the user to get a better overview of the terrain. We have to click outside the island in order to zoom, but always away from the coastline. This is because if the user clicks too close to the coastline, the application will interpret that the user wants to create or modify a hill.

4.2 3D Modeling Operations

In our application, the 3D window shows a volumetric view of the terrain. The user will be able to click with the right button on any of the previously defined hills and can decide on the height of each hill by dragging the mouse up and down. If we drag upwards, then the height will be positive and we will create a hill, and if we drag downwards the terrain will be a valley. Furthermore, by dragging the mouse left and right, the user will be able to decrease and increase the size of the radius of the selected mountain.

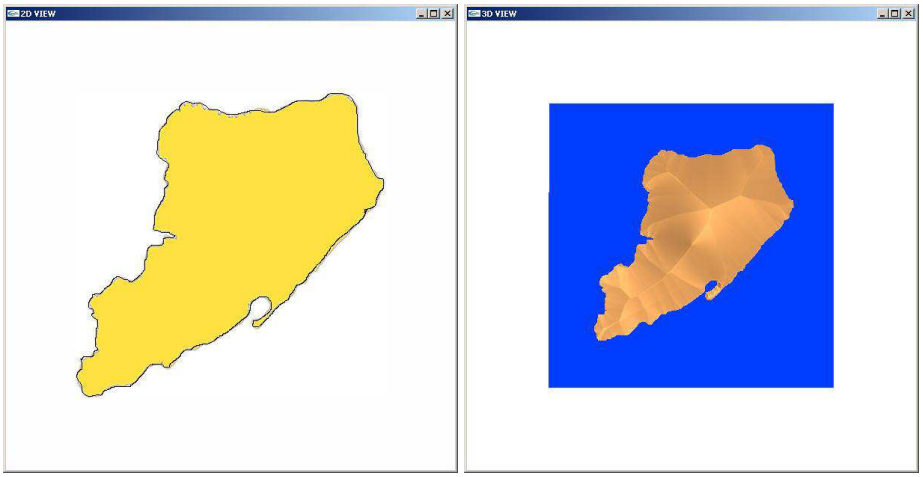
The 3D window also allows for the use of filters, which the user can decide to apply to the whole terrain in order to introduce some fuzzy bumps. Clicking with the left mouse button on any point on the island and dragging upwards will add filters to the terrain. The more we drag upwards, the more bumps are created. On the contrary, if we drag downwards then the application will understand that we want to decrease the number of perturbations.

In addition, the 3D window offers two more functions:

- Clicking with the left button away from any hill and dragging, acts as a rotating function. It allows the island to be rotated inside the window.
- Clicking with the right button away from the terrain and dragging acts as a zoom function, that is to say, this increases or decreases the apparent size of the island.

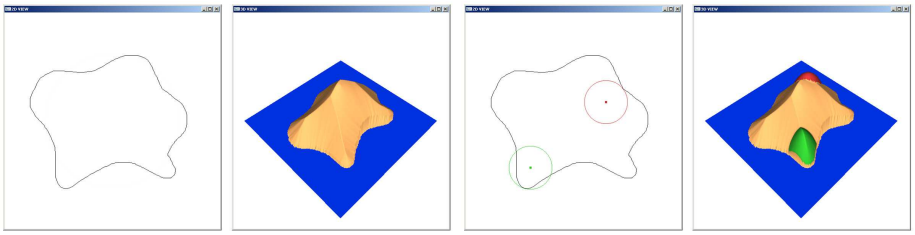
4.3 Contour map

As a guide for the design of the island, it could be possible to use any image as background of the 2D window. Figure 10 presents a recreation of the shoreline



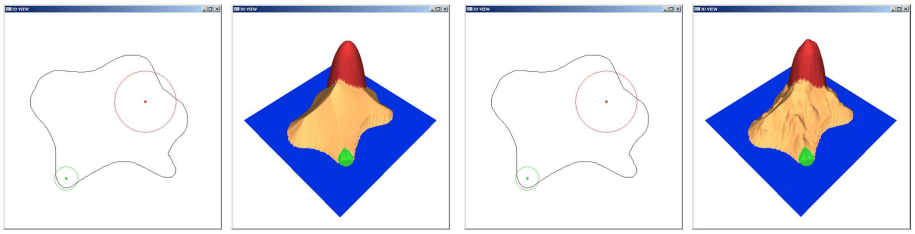
(a) 2D window with the map as background. (b) 3D window.

Fig. 10 Staten Island simulation using an image as a guide.



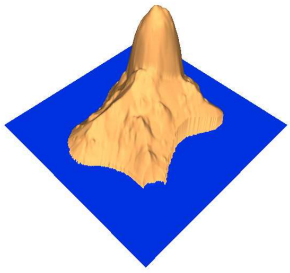
(a) Sketching the coastline.

(b) Defining some hills.



(c) Reshaping and relocating the hills.

(d) Adding some perturbations.



(e) Obtained terrain.

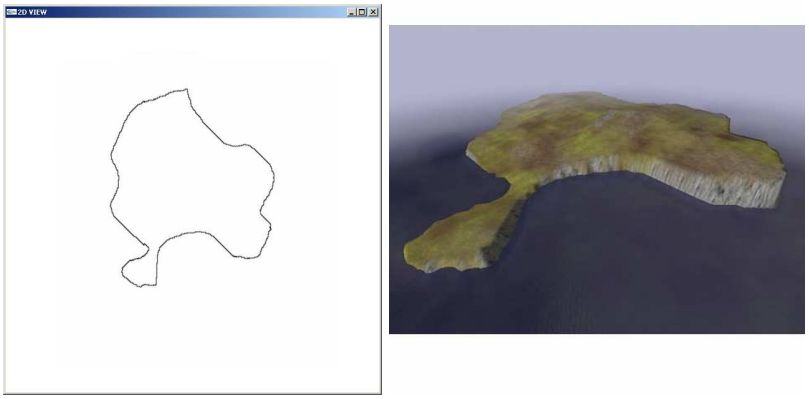
Fig. 11 Designing a sample terrain in four steps.

of Staten Island (New York). Similarly, it could be possible to add hills to those locations indicated by the image in a contour map.

4.4 Assembling all together

The set of operations described above enables the users to design an island with the shape an features that they desire. All the aforementioned functions can be applied in any order, as the terrain update process is capable of reacting properly to any action performed.

In order to exemplify the usage of our application, in Figure 11 we describe a step-by-step design of an island using our framework. Thus, we start from the initial terrain showed in Figure 9 and we draw the coastline that we desire. Then, we locate some hills around the island and we modify their radii, height and location. Then, we apply some filters in order to alter the regular surface of the



(a) 2D Window displaying a sample island.

(b) Game engine rendering a sample island.

Fig. 12 Island output and rendered in a game engine.

terrain. With these simple steps we are able to obtain a terrain which is adequate to be imported into any 3D application that supports heightmaps.

4.5 Output

Our implementation allows the user to save the heightmap in order to be able to use it in different applications, such as game engines and virtual reality applications. Figure 12 presents the 2D window displaying an island and its visualization in a selected game engine.

5 Detailed Implementation

In this section we will describe in detail the different data structures and processes that make up our framework. The user interface was programmed using the GLUT (OpenGL Utility Toolkit) library, offering a simple windowing application programming interface for OpenGL. The interaction with the user was accomplished by applying the methods that this library offers. Nevertheless, at this point mention should be made of the technique used to select the hills in the 3D Window. If the user interacts with the 3D Window, the application renders the contents of this window using the GL_SELECT rasterization mode. This mode, combined with the use of a small frustum located around the clicking point, offers a simple yet efficient method to find out which of the hills has been selected. More precisely, the name of the selected primitive is obtained by reading the contents of the *Selection Buffer* used in this technique, as this buffer stores the information of the objects *rendered* in the picking area.

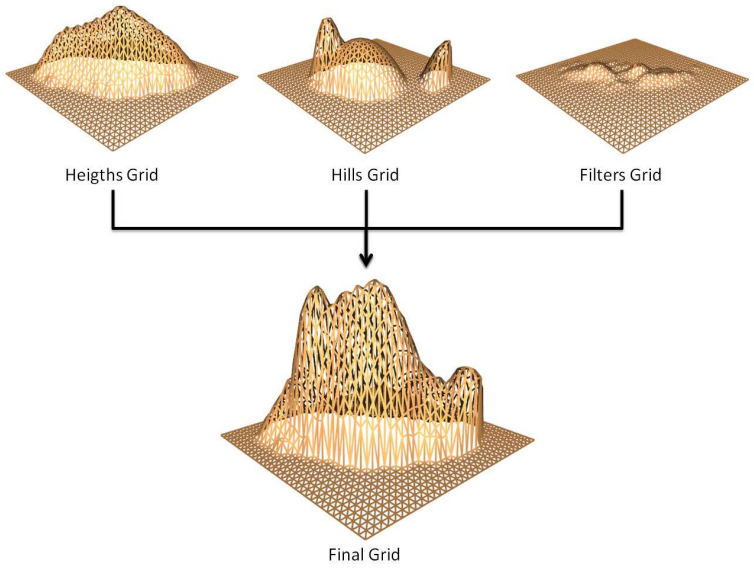


Fig. 13 Composition of the *Final* grid of heights obtained by adding the previously updated grids.

5.1 Data Structures

The final terrain that will be visualized or output is composed of three heightmaps that will be added one after the other. As a consequence, with these three data structures we obtain the *Final Grid*. These heightmaps are stored as grids (2D matrix) of floats:

- *Heights Grid*, which contains the heights of the terrain obtained after modifying the coastline.
- *Hills Grid*, which stores the increments or decrements in height, due to the hill volumes.
- *Filters Grid*, which holds the variations introduced by the inserted filters.
- *Final Grid*, which stores the sum of the three previous grids.

We assume that the *Heights Grid* defines the basic features of the terrain. Then, the other data structures will add more details to the terrain. It is important to note that the division of the terrain information into those three data structures simplifies the process of updating any of them. Thus, for example, adding a hill only involves modifying the *Hills* grid.

In Figure 13 we can see an example of the different grids that make up the *Final Grid*. The *Heights Grid* is obtained after defining the coastline of the island. The *Hills Grid* contains three different hills. Lastly, the *Filters Grid* stores the perturbations introduced by the user. Consequently, these three grids combined together give form to the final terrain. Moreover, we also need other auxiliary data structures, such as:

- *Coastline Vector*, which contains the set of points that form the silhouette of the island.

- *HillPoints Vector*, which stores the position, the height and the radius of each hill.
- *FilterPoints Vector*, which holds the position, the height and the radius of the different perturbations.
- *Vertices Vector*, which includes the vertices information for rendering the final terrain.
- *Indices Vector*, which stores the indices information for rendering the final terrain.

This representation uses an internal triangle mesh to represent the 3D island. The 2D heightmap is linked with the 3D representation in order to allow fast and efficient updates. The implementation has been optimized to ensure that each modeling operation entails updating the minimum amount of information, including both the heightmap and the vertices information, that is to say: spatial coordinates, normals, colors, etc.

5.2 Terrain Algorithm

In Section 4 we presented the different operations that can be applied by the user. In this Section we will thoroughly describe the most important parts of the terrain algorithm.

When the user interacts with the silhouette of the island, the *Coastline Vector* and the *Heights Grid* have to be updated accordingly. If the user creates a *new* isolated coastline then the *Coastline Vector* is completely updated and the *old* coastline is discarded. If the user reshapes an existing coastline then the application has to calculate the intersection points. Depending on the type of operation (defined in Section 4) that the user has performed, the algorithm will select the proper way to combine the *old* and the *new* coastlines.

Once the new *Coastline Vector* has been modified, the algorithm has to update the *Heights Grid* properly. All the points on this grid have to be updated, so that all the ones that are out of the coastline have a zero height while those points which are *onshore* will have a height value calculated following the method presented in Section 3.2.

When the user creates or modifies a hill, the *HillPoints Vector* is updated with the new values. Again, the *Hills Grid* has to be properly modified to follow these changes. For each hill, the algorithm calculates its area of influence. Within that area, all those points which are onshore will have a height calculated by following Equation 3. All those points influenced by more than one hill will add the height values that belong to each hill.

The filtering process is similar to the addition of hills. The only difference is that instead of creating hills by following the user input, the algorithms will give each perturbation stochastic positions, heights and radii. In this case, the more the user drags de mouse upwards or downwards, the more the perturbations will be added or removed.

Algorithm 1 presents a pseudo-code that represents the main operations that are performed to process the terrain. The main advantage of having different structures to store the heights, hills and filters is that we are able to modify each of them separately. In this way, the user can alter a hill position knowing that the rest of the elements will not be modified.

```
switch operationType do
    case shilouetteUpdate
        oldShilouette=currentShilouette;
        currentShilouette=shilouetteMerge(oldShilouette, newShilouette);
        updateHeightsGrid();
        updateFinalGrid(HeightsGrid);
    case hillAdded
        calculateInfluenceArea(newHill);
        updateHillsGrid();
        updateFinalGrid(HillsGrid);
    case hillsUpdated
        forall the hill in hillsUpdated do
            reCalculateInfluenceArea(hill);
            updateHillsGrid();
        end
        updateFinalGrid(HillsGrid);
    case noiseUpdated
        if numPerturbations > 0 then
            addNoise(numPerturbations);
        else
            removeNoise(numPerturbations);
        end
        updateFiltersGrid();
        updateFinalGrid(FiltersGrid);
endsw
renderTerrain(FinalGrid);
```

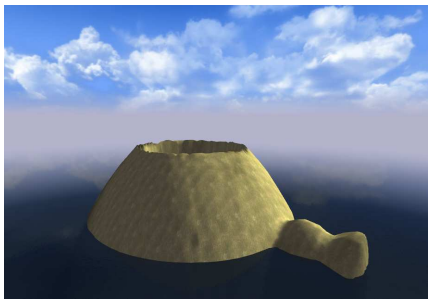
Algorithm 1: Pseudocode of the terrain generation process

6 Results

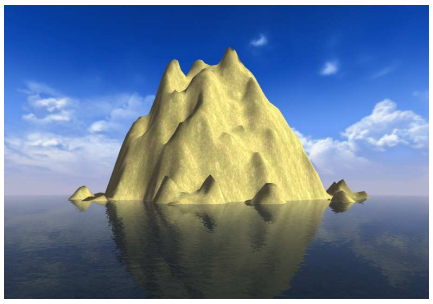
This section presents the results obtained in our tests where we wanted to analyze our sketching application from different perspectives. Thus, we include a comparison against previous solutions as well as a user study in order to assess the validity of our proposal. Moreover, we also studied visual quality and computational cost to prove the performance of our solution. All the tests were performed using a Pentium D 2.8 GHz processor with 2 GB. of RAM and an nVidia GeForce 8800 GT graphics card. The framework was implemented in C++ with OpenGL. It is important to note that the sample images that we have shown throughout the paper were performed with shiny colors that lack realism but clarify the process that we are explaining.

6.1 Visual Results

In order to show the possibilities of our framework, we exported different heightmaps obtained from our terrain generation algorithm. Then we introduced those heightmaps as input into a game engine and we obtained several examples of islands. The game



(a) Volcano-shaped island.



(b) Craggy island obtained after applying many filters to an initially round island.

Fig. 14 Islands created with our algorithm and included in the Torque Game Engine.

engine that we selected to render our islands in our tests is the Torque Game Engine Advanced (TGEA) [11,18] released by GarageGames.

At the beginning of this article, Figure 2 depicted an elongated island. Initially we set up just one big hill on one side of the island and a small one on the other side of the island. Finally, the filters added some perturbations to the terrain.

In Figure 14(a) we have rendered a volcano with two tiny hills on one side. The volcano consists of a big hill but with a hole in the middle. This hole is performed by applying a slightly smaller negative-height hill, located in the middle of the first hill we created. For Figure 14(b) we created a more circular island. This island is also very craggy after having applied many filters. At first, we created a single huge round hill. After that, we included a great number of perturbations until we obtained the desired appearance for the island.

6.2 Performance Comparison

In Section 2.3 we presented a qualitative study of the previously-existing tools for sketching terrain and also of our proposal. In this study we characterized, among other aspects, the complexity of use, the features controlled and the amount of control given to the user over the sketched terrain. Nevertheless, a more quantitative analysis would be interesting in order to clarify the strengths in terrain shape modeling of the tool proposed in this paper.

Among the existing terrain applications presented in the qualitative study we have selected two tools: the one presented by Watanabe and Igarashi [36] and the method proposed by Gain et al. [10]. We have chosen these two methods as they are the ones which are strictly more oriented toward offering a complete sketching interface for modeling terrain. Figure 15 presents a visual comparison where two terrains obtained with these tools are simulated with our tool by an expert user. The terrain presented in Figure 15(a) can be simulated fast, as flat mountains can be easily sketched with our technique. The terrain depicted in Figure 15(c) is much more complex, as it has been modeled by a designer to simulate the Tower Karst Islands. In this second example, the orography has been simulated respecting the mountain locations, heights and general appearance. Nevertheless, the hilltops are

difficult to simulate in our proposal, as it has been designed to give form to an imagined surface and not to a very precise ridge.

From a different perspective, it would also be interesting to consider the time required to model these terrains using the three tools. According to the authors of each tool, the modelers required less than 10 minutes to model the surface in Figure 15(a), while the terrain presented in Figure 15(c) involved less than 30 minutes. The terrain simulations performed using our approach supposed less than 2 minutes for the former and about 15 minutes for the latter. These time reductions are due to the fact that our approach is capable of offering sufficient control over the terrain without resorting to a complex interface which can result in an increase of the time required to obtain the desired appearance.

The complexity of the interface can also be used to compare different tools, considering how many operations can the user choose to model the terrain surface (other operations like noise, adding decorative elements or defining the coastline are not considered to offer a fair comparison). In this sense, the first tool offered a very simple interface where only one operation to define the ridge-line was offered. The second tool involved more than 7 operations to sketch the ridge-line, to define the width of the mountains or to manage the occlusions among mountains. Our proposal uses 4 operations to define the location, width and height of the hills and to modify the silhouette of the island (which can also affect the terrain surface). Thus, and considering the terrains shown in Figure 15, we can conclude that a balance between complexity and usability must be found to obtain a tool which can be simple yet capable of modeling terrains with good visual appearance.

6.3 User Study

We have considered that it is compulsory to perform a user study in order to evaluate the quality of our sketching application.

One of the first tasks to perform when conducting a usability study is to decide on the persons that we are going to recruit. Different studies have proven that 5 users are enough to assess the quality of software applications [22], although evaluating visualization results need a different amount of volunteers in order to obtain valuable results [17]. As a consequence, we will need more users as we also test their *cognitive* capabilities, which means in our case, the faculty of a user to process information after having perceived the visual inputs.

In our case we have conducted the test to 30 people, grouped in three categories depending on their expertise both in computer use in general and in computer design in particular. Our tests were carried out in our university lab, and all of the participants were staff or students at the university.

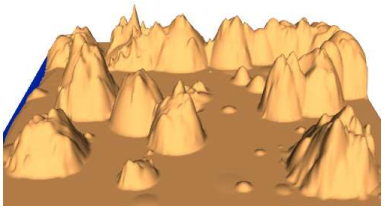
The objective of our informal user study consisted in analyzing our sketching application from different perspectives. First, we asked our volunteers to grade from 1 to 10 the overall quality perception of the application, being 1 the worst and 10 the best. Secondly, we monitorized their activity to record the time passed until they obtained a satisfactory terrain. Finally, we asked them to report any difficulty or mistake they could find in using our application.

The test sessions consisted in giving the first-time users some basic indications in order to make them know how our application works. After that, we made them try to draw an island silhouette with the desired hills. They tried different

m

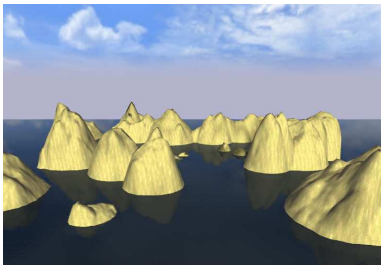
(a) Terrain authored using the tool presented in [36].

(b) Similar terrain simulated with our approach.



(c) Terrain authored using the tool presented in [10].

(d) Similar terrain simulated with our approach.



(e) Terrain shown in Figure 15(d) visualized in the Torque Game Engine.

Fig. 15 Image comparison against two previous sketching terrain applications.

Study Group	Number of Volunteers	Overall Satisfaction	Avg. Required Time (min.)	Observed Problems
Computer Scientists	12	8	3	12
Designers	7	6	5	7
Other Disciplines	11	9	4	2

Table 2 Results obtained with 30 university volunteers.

coastline, hills and filters until they outlined the desired appearance of the island and the terrain.

In Table 2 we present the average results obtained with university volunteers. Most of them were satisfied with their results after less than five minutes. It is important to mention that both computer scientist and designers found several problems that helped us to improve the application. Most of them found the ap-

plication difficult to use at the beginning, although after some practice they started modeling their terrains. These initial usability problems helped us to modify some aspects of the application and also encouraged us to create a brief guide to explain how the application works. Moreover, most users found the application *funny* and tried the different possibilities after acquiring a little expertise. Finally, we also encouraged some of them to include the terrain inside the Torque Game Engine [11,18] with our help, in order to give them the possibility to experience gaming over their modeled island and terrain.

6.4 Storage and Computational Costs

Finally, the tests would not be complete without a time/space complexity analysis.

6.4.1 Storage Cost

It would be interesting to make an estimation of the storage cost of each of the data structures presented in Section 5, supposing that the cost of an integer and a float is a word (4 bytes).

- *Vertices Vector*. This data structure stores normals, colors and positions for each vertex. Being v the amount of vertices, the cost is $2v + 2v + 3v = 8v$.
- *Indices Vector*. For each triangle we need 3 words to store the indices of the 3 vertices that form it. Being f the number of faces, the storage cost is $3f$.
- *Coastline Vector*. The cost of this data structure depends on the resolution of the grid nxn. In our tests a vector of size $10n$ is sufficient to store the coastline.
- *HillPoints Vector*. The cost of this structure is modified dynamically depending on the amount of hills employed h. The cost can be calculated as $4h$, as we need 2 words for the position, 1 for the height and 1 for the radius.
- *FilterPoints Vector*. Similarly to the HillPoints Vector, the size of this data structure depends on the amount of perturbations applied p, and the storage cost can be calculated as $4p$.

With this information, the storage cost of our solution can be expressed as:

$$8v + 3f + 10n + 4h + 4p \tag{4}$$

The amount of vertices v that are needed to represent a grid of size nxn is n^2. Moreover, the amount of triangular faces f that are needed to index those vertices is $(n-1)^2 * 2$. With this information, the storage cost can be calculated using Equation 5. In this equation the cost of the Coastline, HillPoints and FilterPoints vectors has been neglected because, as we will see, their size is very small compared to the vertices and indices cost.

$$8n^2 + 3((n-1)^2 * 2) + 10n + 4h + 4p \cong 8n^2 + 6n^2 \cong 14n^2 \tag{5}$$

With the equation presented above, we can conclude that the storage cost of a terrain grid of size nxn is $14n^2$ words, which is $56n^2$ bytes. Table 3 presents a real study of the storage cost of our proposal with different grid resolutions.

Grid size	Vertices number	Triangles number	Storage Cost (Kb.)			
			Vertices	Triangles	Coastline	Total
32x32	1,024	1,922	32.00	22.52	1.25	55.77
64x64	4,096	7,938	128.00	93.02	2.50	223.52
128x128	16,384	32,258	512.00	378.02	5.00	895.02
256x256	65,536	130,050	2,048.00	1,524.02	10.00	3,582.02
512x512	262,144	522,242	8,192.00	6,120.02	20.00	14,322.02
1,024x1,024	1,048,576	2,093,058	32,768.00	24,528.02	40.00	57,336.02

Table 3 Storage cost of our terrain generation algorithm using different grid resolutions.

From these results we can see how the largest amount of memory is needed for storage vertices and indices information. We must remember that we store, for each vertex, its spatial coordinates as well as its color and normal information. We could reduce the storage cost of the vertices by calculating the color and the normal directly in the programmable units of the Graphics Card. It can also be seen that for a grid resolution of *1,024x1,024* the memory needed is greater than 50 Mb.; this is a high value but we must also consider that it also involves dealing with more than 1 million vertices, which is too much considering the scope of our sketching terrain proposal. It is worth mentioning that the terrains shown in this paper have been generated using a *512x512* grid, which is also the case of the terrains presented in [10].

Regarding the storage cost presented in Equation 5, the obtained estimation is very accurate. For example, for a grid resolution of *256x256* the actual cost is 3,582.02 Kb., while using Equation 5 we would obtain 3,584 Kbytes. As we mentioned previously, for this estimation we neglected the cost of the Coastline, HillPoints and FilterPoints vectors. The value of the Coastline cost has been included in Table 3, although it represents a very small percentage of the storage cost. The data structure needed to store a hill or a perturbation has a 0.015625 Kb. cost. Thus, having 50 hills and 1,000 perturbations would only involve about 16 Kb., which for a grid resolution greater than *256x256* is a very small value.

6.4.2 Computational Cost

To evaluate the time requisites of the presented solution we have conducted several tests to measure the performance of the different operations that can be used to alter the terrain. Table 4 offers the time needed to modify the terrain when updating the hills, the noise or the silhouette of the terrain. As we did when studying the storage cost, the results are offered for different grid resolutions.

From these results we can see how altering hills or adding noise does not involve a high computational cost, as the amount of points to modify in either case is limited. This is not the case of the silhouette update. In Section 3.2 we presented the algorithm applied to re-calculate the terrain when modifying the silhouette, indicating that all the points included in the new silhouette must be processed to calculate the new height. In this test we have considered the worst case possible where the new silhouette covers the entire grid and, thus, all vertices must be processed. This is the reason why the time needed for this operation is very high, as for the *1,024x1,024* grid we must check more than 1 million points using Equations 1 and 2. As we have already commented, this is the worst case

Grid size	hillsUpdated (50 hills)	noiseUpdated (1,000 perturbations)	shilouetteUpdate
32x32	0.001	0.001	0.007
64x64	0.002	0.002	0.030
128x128	0.002	0.002	0.13
256x256	0.003	0.003	0.58
512x512	0.005	0.007	1.94
1,024x1,024	0.047	0.036	8.13

Table 4 Computational cost of our terrain generation algorithm (in seconds).

possible and in a real execution of our application the amount of points to process would be smaller. In addition, the silhouette update must only be executed when the user actually alters the silhouette and, thus, while modifying the hills or adding noise this costlier process is not performed.

7 Conclusions

This paper presents an algorithm for terrain generation which is suitable for users who wish to have full control over the whole creation process, offering a balance between high user control and low complexity of use. We have also presented a simple tool for creating solid models of imaginary islands. The tool is easy to use and requires only a minimal user interface, with all of its major operations being controlled by a two-button mouse. From this application, the user can add, remove and reshape existing hills interactively and the terrain will be updated accordingly. Moreover, the user is able to modify the silhouette of the island and add fuzzy bumps as desired. The use of two different windows to perform separately 2D and 3D operations simplifies the interface if compared with previous solutions.

Accordingly, the images of islands that we have included in the previous section show how our approach is capable of offering terrains adapted to the needs of the users. Thus, the end user can decide on the final appearance of the island, as it is possible to apply any number of filters. Nevertheless, the user could choose not to apply filters in order to obtain a fairly rounded terrain which could be useful for a cartoon-like environment. The usability study, which was performed among people with different levels of computer skills, showed that the user interface we finally selected was comfortable and adequate in most cases. We must note that our tool could be equally applied to model general terrain surfaces, and the images shown in Figure 15 show how we can simulate terrains generated with other sketching tools. From a different perspective, from the experiments we can also conclude that our solution has competitive storage and computational costs.

The most promising area for future work consists in adding new features to our existing application. We note that increasing the features is, in principle, easy. For example, it would be straightforward to change the application so that the designer could mark particular areas of the island as beach or forest, texture them as desired and include vegetation and other decorative elements. The problem with any of these is that each additional option would make the user interface more complicated, thereby losing a major advantage of the existing user interface, its simplicity. Nevertheless, a more complicated user interface could be justified

by analogy with sketching on paper, as paper maps often use different colours of ink for forests and lakes. A group of new features which could be added without compromising the user interface is simulation of physical processes. For example, the height contours of an island could be determined, not by a convenient trigonometrical formula, but by the way the lava could flow after a real volcanic eruption and the way that the resulting shape could be sculpted by wind and rainfall.

Moreover, we would also like to analyze the possibility of using different shapes for defining the hills and also different noise functions. The use of elliptic paraboloids has proven to be adequate for our purposes, but we would like to enhance the terrains generated without increasing the complexity of use of the application.

Acknowledgements

This work has been funded by the Spanish Ministry of Science and Technology (projects TIN2010-21089-C03-03, TSI-020400-2009-0133 and DPI2011-28507-C02-02), by the European Union (ITEA2 IP08009), by the Ramon y Cajal Scholarship Programme and by FEDER funds.

References

1. Anastacio, F., Sousa, M., Samavati, F., Jorge, J.: Modeling plant structures using concept sketches. In: Proc. of the Int. Symposium on Non-photorealistic animation and rendering, pp. 105–113 (2006)
2. Belhadj, F.: Terrain modeling: a constrained fractal model. In: Int. Conf. on Computer Graphics, virtual reality, visualisation and interaction in Africa, pp. 197–204 (2007)
3. Cao, L., Liu, J., Tang, X.: What the back of the object looks like: 3D reconstruction from line drawings without hidden lines. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(3), 507–517 (2008)
4. Cohen, J., Hughes, J., Zeleznik, R.: Harold: a world made of drawings. In: Int. Symposium on Non Photorealistic Animation and Rendering (NPAR), pp. 83–90 (2000)
5. Cook, M., Agah, A.: A survey of sketch-based 3-d modeling techniques. Interact. Comput. **21**(3), 201–211 (2009)
6. Dachsbacher, C.: Interactive terrain rendering: Towards realism with procedural models and graphics hardware. Ph.D. thesis (2006)
7. Digital Element: Worldbuilder. http://www.digi-element.com/wb/ [accessed 29 April, 2011] (2006)
8. Fefilatyev, S., Smarodzinava, V., Hall, L., Goldgof, D.: Horizon detection using machine learning techniques. In: 5th International Conference on Machine Learning and Applications, pp. 17–21 (2006)
9. Frade, M., de Vega, F.F., Cotta, C.: Breeding terrains with genetic terrain programming: The evolution of terrain generators. International Journal of Computer Games Technology **2009** (2009)
10. Gain, J., Marais, P., Strasser, W.: Terrain sketching. In: Proc. of the 2009 Symp. on Interactive 3D graphics and games, pp. 31–38 (2009)
11. Garage Games: Torque game engine advanced. http://www.garagegames.com/ [accessed 29 April, 2011] (2008)
12. Hoiem, D., Efros, A., Hebert, M.: Automatic photo pop-up. In: Siggraph '05: ACM Siggraph Papers, pp. 577–584. ACM (2005)
13. Hormann, K., Spinello, S., Schroder, P.: c^1-continuous terrain reconstruction from sparse contours. In: Proc. of Int. Workshop Vision, Modeling, and Visualization, pp. 289–297 (2003)
14. Inc., A.: Autodesk 3ds max. http://usa.autodesk.com/adsk/ (2010)
15. Kara, L., Shimada, K.: Sketch-based design of 3D geometry. In: Eurographics Workshop on Sketch-Based Modelling, pp. 59–66 (2006)

16. Kelley, A., Malin, M., Nielson, G.: Terrain simulation using a model of stream erosion. In: Siggraph: 15th annual conference on Computer graphics and interactive techniques, pp. 263–268 (1988)
17. Kosara, R., Healey, C., Interrante, V., Laidlaw, D., Ware, C.: User studies: Why, how, and when? Ieee Computer Graphics & Applications **23**(4), 20–25 (2003)
18. Maurina, E.: The Game Programmer's Guide to Torque. AK Peters, Ltd, 608 pp. (2006)
19. Mori, Y., Igarashi, T.: Plushie: an interactive design system for plush toys. In: Siggraph '07: ACM Siggraph papers, p. 45 (2007)
20. Musgrave, F., Kolb, C., Mace, R.: The synthesis and rendering of eroded fractal terrains. In: Siggraph'89, pp. 41–50 (1989)
21. Neidhold, B., Wacker, M., Deussen, O.: Interactive physically based fluid and erosion simulation. In: Eurographics Workshop on Natural Phenomena, pp. 25–32 (2006)
22. Nielsen, J., Landauer, T.: A mathematical model of the finding of usability problems. In: Proc. of the INTERACT '93 and CHI '93 conference on Human factors in computing systems, pp. 206–213 (1993)
23. Olsen, L., Samavati, F., Sousa, M., Jorge, J.: Sketch-based modeling: A survey. Computers & Graphics **33**(1), 85 – 103 (2009)
24. Ong, T., Saunders, R., Keyser, J., Leggett, J.: Terrain generation using genetic algorithms. In: GECCO'05: Proc. of the 2005 conference on Genetic and evolutionary computation, pp. 1463–1470 (2005)
25. Owada, S., Nielsen, F., Nakazawa, K., Igarashi, T.: A sketching interface for modeling the internal structures of 3D shapes. In: SIGGRAPH '07: ACM SIGGRAPH 2007 courses, p. 38 (2007)
26. Planetside: Terragen. http://www.planetside.co.uk/terragen/ [accessed 29 April, 2011] (2008)
27. Quad: Grome. http://www.quadsoftware.com/ [accessed 29 April, 2011] (2008)
28. Rusnell, B., Mould, D., Eramian, M.: Feature-rich distance-based terrain synthesis. Visual Computer **25**, 573–579 (2009)
29. Schmitt, S.: World machine. http://www.world-machine.com/about.html [accessed 29 April, 2011] (2006)
30. Schneider, J., Boldte, T., Westermann, R.: Real-time editing, synthesis, and rendering of infinite landscapes on GPUs. In: Vision, Modeling and Visualization 2006, pp. 145–152 (2006)
31. Stachowski, K.: Terraineer. http://terraineer.sourceforge.net/ [accessed 29 April, 2011] (2006)
32. Teoh, S.T.: Riverland: An efficient procedural modeling system for creating realistic-looking terrains. In: Proc. of the 5th International Symposium on Visual Computing (ISVC'09), pp. 468–479 (2009)
33. Torpy, A.: Large 3D terrain generator (l3dt). http://www.bundysoft.com/L3DT/ [accessed 29 April, 2011] (2009)
34. United States Geological Survey: National mapping program standards. http://rockyweb.cr.usgs.gov/nmpstds/demstds.html [accessed 29 April, 2011] (2003)
35. Varley, P., Chover, M., Puig-Centelles, A.: Sketching islands for a game environment. In: 5th European Conference on Visual Media Production (CVMP), pp. 1–10 (2008)
36. Watanabe, N., Igarashi, T.: A sketching interface for terrain modeling. In: SIGGRAPH '04: ACM SIGGRAPH 2004 Posters, p. 73 (2004)
37. Wind Erosion Research Unit, Kansas State University: Weru. wind erosion simulation models. http://www.weru.ksu.edu/weps/wepshome.html (2003)
38. Wither, J., Bouthors, A., Cani, M.: Rapid sketch modeling of clouds. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling (2008)
39. Worboys, M.F.: GIS: A Computing Perspective. Taylor and Francis, 373 pp (1995)
40. Wünsche, B., Keymer, D., Amor, R.: Sketch, click, plug and play: accelerated design of virtual environments by integrating multimedia and sketch content into game engines. In: Proc. of the 11th International Conference of the NZ Chapter of the ACM Special Interest Group on Human-Computer Interaction (CHINZ'10) (2010)
41. Zhou, H., Sun, J., Turk, G., Rehg, J.: Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics **13**(4), 834–848 (2007)